AI Wall Building in *Empire Earth® II*

Tara Teich, Dr. Ian Lane Davis

Mad Doc Software
100 Burtt Rd, Suite 220
Andover, MA 01810

Abstract

Real-Time Strategy games are among the most popular genres of commercial PC games, and also have widely applicable analogs in the field of "Serious Games" such as military simulations, city planning, and other forms of simulation involving multi-agent coordination and an underlying economy. One of the core tasks in playing a traditional Real-Time Strategy game is building a base in an effective manner and defending it well. Creating an AI that can construct a successful wall was one of the more challenging areas of development on *Empire Earth® II*, as building a wall requires analysis of the terrain and techniques from computational geometry. An effective wall can hold off enemy troops and keep battles away from the delicate economy inside the base.

Introduction

The ultimate goal of the AI in a Real-Time Strategy (RTS) game is to put up a good fight and seem to be a human opponent. To this end, the AI in *Empire Earth® II* (EE2) constructs bases of buildings in tight clusters and seeks to defend these bases by enclosing them with a wall for protection. Starting with a very basic initial plan, we attempted to create a system that could dynamically alter its plans as obstacles cropped up and use a modified convex hull algorithm to maintain a smooth and minimal wall plan. Interesting challenges arose from taking this simple algorithm from paper into a constantly changing, unpredictable environment.

Walls

Walls are static defenses that surround bases to slow down attackers. They are invaluable in protecting from quick raids against a player's citizenry, as the opponent will need a larger force to breach the walls, and will allow for advance warning of the impending attack. The ideal wall will surround as much of a base as possible, and will take advantage of terrain to provide obstacles where possible, so as to only build the minimal wall sections…but at the same time, it will leave some room for growth within the walls. The EE2 AI does not build ideal walls (due to the inherent complexities of even defining optimal), but instead attempts to wall in the base at a tight radius around its starting layout. Our goal is to make a tight wall so as to minimize costs in resources and time to construct.

We start with a plan of a box at a fixed distance from the starting base location. The algorithm steps along each edge of the plan and checks if the edge is valid. Intersections with terrain elements are considered valid, intersections with other buildings are not. If the edge is invalid, it takes the location of the obstacle intersection and creates a new edge node by stepping out 1/2 the length of the original edge on the normal pointing away from the interior of the plan. The original edge is removed and two new edges are created between the original end nodes and the newly created node.

This process is repeated until all edges are valid. We are left with a very jaggy wall plan. We need to smooth it out, so we find the convex hull for our edge nodes. We chose the Graham Scan (Sedgewick 1992) method. For our initial wall plan, this yields very favorable results.

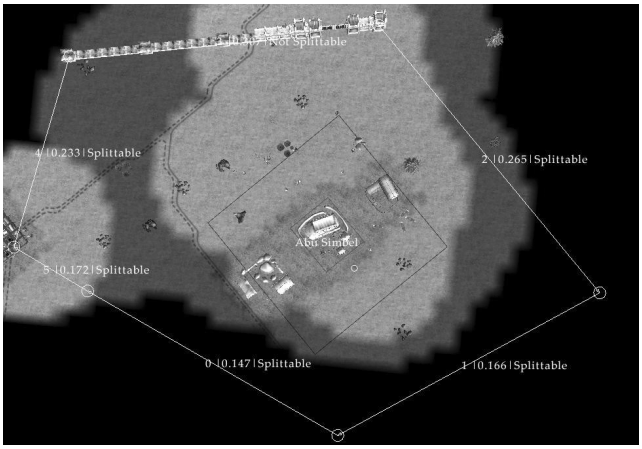

Figure 1. Partially built wall plan

In Figure 1, the AI has just begun its wall plan. One edge is under construction, and the rest are shown via debug displays. Edges marked splittable can be split by the method described above.

Walls tend to be very expensive and also require a high commitment of citizens to build them. We build the wall one edge at a time to help spread out the cost and workforce burden. The result of this was that walls also tend to take a very long time to be completed and buildings

were sometimes placed smack in the midst of a planned wall edge.

Fortunately, we can use our smoothing algorithm to compensate for this. Before building an edge, the wall builder checks its validity. If it is no longer valid, it will repeat the process it used at creation, splitting the edge and ultimately re-smoothing the graph. This results in vast complications as we can have a partially built wall. This means that the convex hull algorithm cannot treat all nodes equally, as some of them are not removable.

The first part of the Graham Scan algorithm involves finding the bottom most node and then sorting all nodes by their rotation from that point. Because some edges are already built, we need to fix up the order after the sort is complete. We loop through all edges and for each edge that is already built we make sure that the tail node immediately succeeds the head node. This is further complicated by the possibility that multiple adjacent edges are already completed, so nodes must be moved to ensure that all connections are maintained. Following is the algorithm for the Graham Scan with our additions in bold.

1. Find bottom most edge node
2. Sort all nodes at their angle from this minimum point
3. **Loop through all edges in wall plan**
4. **If edge is already built, move the two nodes of the edge adjacent in the sorted list**
5. Loop through sorted node list
6. Look at nodes in groups of three
7. while the angle between these nodes is counter-clockwise **and the middle node is not on a started edge** drop the middle node and use last node and two nodes before the middle as the group of three

We do not want to remove any nodes that are part of an existing edge. Our AI never destroys wall segments it has already built as this is both a waste of money and would make a player who observes the behavior think the AI was not being smart (in commercial games these perceptions are significant). Unfortunately this yields some non-optimal wall layouts when new obstacles make existing edges lie on the interior of the convex hull.

Roads pose a particular challenge to wall plans. Because a road is a continuous obstacle, the usual solution of stepping away from the obstacle doesn't work unless the road and wall are nearly parallel. We needed to add a special case to deal with this intersection.

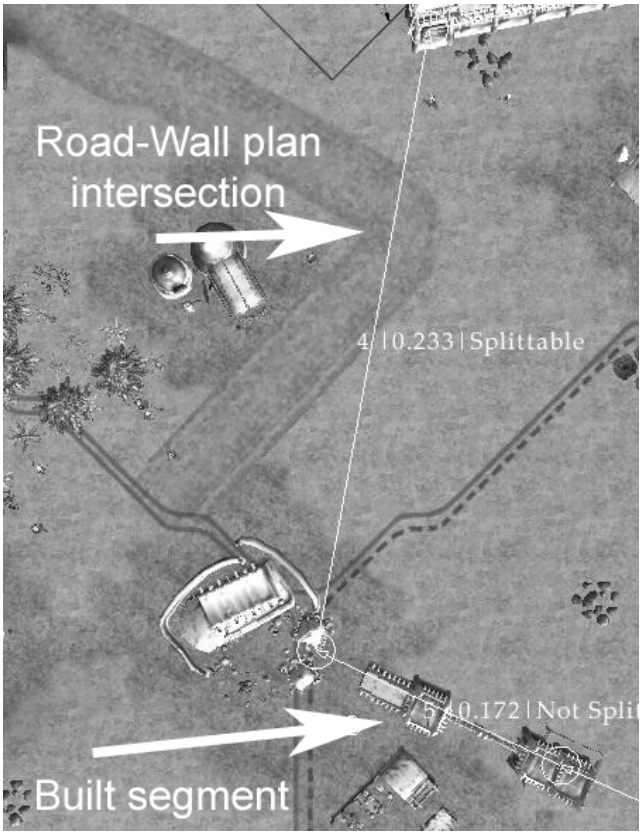

Figure 2. Wall plan with some built segments and a road *intersecting the plan.*

The only allowable intersections between roads and walls are at a perpendicular, which causes the wall system to automatically insert a gate. The goal of the wall planner therefore becomes to insert a segment perpendicular to the road and connect the ends of the segment to the existing wall plan. In most cases, this works well.

Figures 2 and 3 show the wall from Figure 1 in a further state of development. A road is intruding on the leftmost portion of the plan, and the edges connecting to this invalid edge are both already constructed. The AI manages to successfully create a gate edge, visible in the upper left section of the completed wall in Figure 3. Unfortunately, this wall has two problems: a node in the lower right is partially overlapping a building, and the plan doesn't handle the small forest very well. The node inside the building cannot be moved because that edge is already constructed. Other than this problem area, the wall is successful - it almost completely encloses the base.

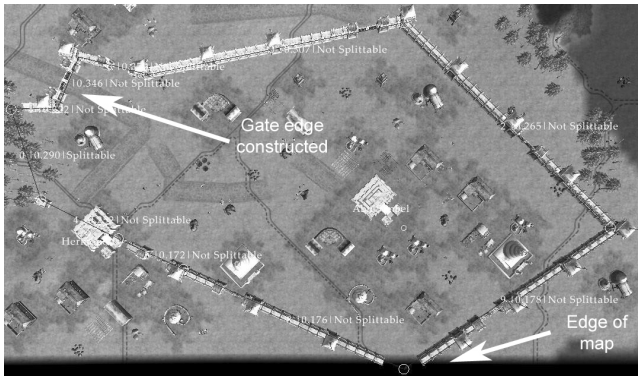

Figure 3. Complete wall with inserted gate

Close examination of the left hand side of Figure 3 reveals the problem. The lower left area is densely packed with buildings and terrain elements such as trees. Given our requirement that the AI never destroy an existing structure, there is no clear solution to this situation.

Future Work

The best results in crafting RTS AI usually come from studying what human players do and attempting to mimic their behavior. Our initial algorithm design was flawed, as it was not derived from a system that a human would use. Human players do a high level terrain analysis and lay out a wall plan that encompasses more territory than they currently control so as to provide room to grow. The *Empire Earth® II* AI has no notion of where things will go, only where they are. Humans will also identify permanent terrain features that make effect barriers and incorporate these into their wall plans. The EE2 AI makes little use of terrain features to minimize the amount of wall to build, and doesn't necessarily realize when a wall is spread over non-contiguous areas, such as when the wall crosses a river. A more predictive system could avoid some of these pitfalls, though the predications can be complex, especially when multiple players are involved but even within one team's AI as you cannot be sure when the overall plan will change as the current battle conditions change.

In future work on wall building, we would like to start with a better initial wall plan. Beginning with an analysis phase that could identify terrain features to incorporate into the wall plan would yield walls that require fewer edges and therefore cost fewer resources and can be completed more rapidly. We could then apply the techniques described here to handle obstacles in planned edges as well as a modification of the smoothing algorithm to treat terrain features as non-removable edges in the graph and ensure the entire plan remains as close to a convex form as possible.

Conclusions

AI wall building is a challenging problem that has received very little industry attention. Many top-tier RTS titles avoid the problem entirely. We examined recent titles such as *Age of Empires® III* and never saw the AI players attempt to construct a wall. Ultimately, our goal is to create a challenging opponent for the human player that enables an enjoyable gameplay experience and appears as human as possible. Building a wall is a complex problem of computational geometry that is only a small part of contributing to this overall goal, yet takes a significant contribution from the developer. As developers place more attention on the AI in general, more focus will be placed on walls in particular. Another current title, *The Battle for Middle-Earth™ II* does tackle the challenge. The first *Empire Earth®* AI built walls as well, though it simply boxed in its entire corner of the map with no consideration of minimizing wall cost. Future developers entering the arena will benefit from understanding the techniques presented here, and an expansion of this geometric algorithm will create a better, more believable AI.

References

Davis, I. L. 1999. Strategies for Strategy Game AI. *AAAI 1999 Spring Symposium on AI & Computer Games Proceedings*. March, 1999

Davis, I. L. 2000. Warp Speed: Path Planning for Star Trek: Armada. *AAAI 2000 Spring Symposium on Interactive Entertainment and AI Proceedings*. March, 2000.

Sedgewick, R. 1992. *Algorithms in C++*. Reading, Mass.: Addison-Wesley.

Preparata, F. P. and Shamos, M. I. 1985. *Computational Geometry: An Introduction.* Springer-Verlag